% La gestion de version avec GIT % % LTE Consulting

Qu'est-ce que git

- Logiciel de gestion de version distribué
- Un ensemble de petits outils pour manipuler un arbre d'objets

Historique de GIT

Premier essai

Nous allons essayer les commandes les plus basiques puis nous aborderons les choses plus en détail...

Configuration

- git config --global user.name "Mon nom"
- git config --global user.email "mon.email@mail.un"

Créer un repository

- mkdir projet
- cd projet
- git init

Les commandes de base

- git add file1 file1 ...
- git diff --cached
- git status
- git commit

On efface tout et on recommence !

```
cd ..
rm -Rf projet
```

Cloner un repo

Fonctionne avec les protocoles : fichiers, http, git, ssh, ...

```
git clone git://git.kernel.org/pub/scm/git/git.git
```

Vous êtes maintenant "branchés" sur le repository "central"

Revenons à la théorie : le modèle objet

Sha

Les SHA sont partour dans git !

40 caractères représentant la signature d'un contenu

```
6ff87c4664981e4397625791c8ea3bbb5f2279a3
```

Il est **impossible** que deux contenus différents aient le même *SHA*

Ceci apporte certains avantages :

- comparaison d'objets rapide,
- les sha sont identiques sur les repos différents,
- détection des erreurs de cohérence.

Les objets

Un **type**, une **taile** et un **contenu**

Il y a quatre types :

- **BLOB** : stockage d'un fichier
- **TREE** : référence des sous TREE et des BLOBS
- **COMMIT** : pointe vers un TREE et contient des métadonnées (auteur, date, commit(s) parents)
- **TAG** : utilisé pour tagger des commits

Les blobs

Pour montrer le contenu d'un blob : `bash git show 6ff87c4664` - Deux contenus identiques partageront le même blob - Invariant : `blob.nom == sha1( blob.contenu )` - Indépendant de l'emplacement des données

L'objet tree

- C'est une liste de pointeurs vers des **blobs** et des **trees** - Utilisé en général pour représenter un répertoire `bash git show # fonctionne mais il y a mieux ! git ls-tree fb3a8bdd0ce 100644 blob 63c918c667fa005ff12ad89437f2fdc80926e21c .gitignore 040000 tree 2fb783e477100ce076f6bf57e4a6f026013dc745 Documentation 100644 blob 6ff87c4664981e4397625791c8ea3bbb5f2279a3 Makefile` - Deux trees n'ont le même nom que s'ils ont le même contenu, ceci facilite les recherches

L'objet commit

Créé avec la commande git commit Pour explorer : `bash git show HEAD --pretty=raw commit fa7ed850e9d3a102e5e525a1c699fcbe930bbc0e tree d0ae012e7b340891eb30c3b66a9fd8a5c08b6457 parent 3ad3da20084dbab3e49e04773730e3e3dfcb32ee author Arnaud Tournier <ltearno@gmail.com/> 1421416426 +0100 committer Arnaud Tournier <ltearno@gmail.com> 1421416426 +0100 # comment...`

Exemple de contenu

L'objet tag

- Utilisés pour stocker des tags signés - les tags légers sont stockés dans `refs/tags/` Création avec `bash git tag` Pour explorer : `bash git cat-file tag v2.1 object 8e26b5aaf0467cda204e82805e0a0109de7193c2 type commit tag v2.1 tagger Arnaud Tournier <ltearno@gmail.com> 1382445936 +0200 # comment...`

Le répertoire .git

----- | ----- HEAD | pointeur vers votre branche courante config | configuration de vos préférences description | description de votre projet hooks/ | pre/post action hooks index | fichier d'index logs/ | un historique de votre branche objects/ | vos objets (commits, trees, blobs, tags) refs/ | pointeurs vers vos branches

Le répertoire de travail

- C'est votre espace de travail
- A la racine de votre projet
- Il est souvent modifié !

Cycle de vie

L'index

- fichier `.git/index`
- Zone d'assemblage pour construire un commit
- A la création d'un commit, ce n'est pas le répertoire de travail qui est pris en compte, mais cette zone dite de **staging**

Voici quelques commandes associées :

```
git add file
git rm file
git status
git commit
...
```

Les branches

- Ce sont simplement des pointeurs sur des commits !
- Stockées dans le répertoire `.git/ref/`

Quelques commandes :

```
git branch branch_name
git checkout branch_name
```

Illustration

Utilisation avancée

Utilisation des branches

- git branch
- git branch experience
- git checkout experience
- git checkout -b experience
- git checkout master && git merge experience
- git branch -d experience

Les cas de merge

- Pas de conflit, fast-forward
- Pas de conflit, mais il faut un commit de merge
- Conflits !

Résoudre des conflits

```
<<<<<<< HEAD:file.txt
Hello world
=======
Goodbye
>>>>>>> 77976da35a11db4580b80ae27e8d65caf5208086:file.txt
```

Historique

- git log
- git log --since="2 weeks ago"
- git log extract.sh
- git log commons/
- git log -S'foo()'
- git log -p # pour voir les patchs
- git log --stat
- git log --pretty=oneline
- git log --pretty=format:'%h was %an, %ar, message: %s'
- git log --graph

Ordonnancement de l'historique

- git log --pretty=format:'%h : %s' --topo-order --graph

```
* 4a904d7 : Merge branch 'idx2'
|\
| * dfeffce : merged in bryces changes and fixed some testing issues
| |\
| | * 23f4ecf : Clarify how to get a full count out of Repo#commits
| | * 9d6d250 : Appropriate time-zone test fix from halorgium
```

Comparaison de commits

- git diff master..test
- git diff master...test

Etat du repository

- Répertoire de travail : `git diff`
- Index : `git diff --cached`
- Les deux : `git diff HEAD`
- Seulement les stats : `git diff --stat`
- Et ça ? `git diff HEAD -- ./lib`

Workflows

Les tags

Un label qui pointe sur un commit

- git tag v2.3.4b 1b238ae12

Les objets tag

- git tag -a v2.3.4b 1b238ae12

`-s` pour signer le tag, après avoir configuré la clé utilisateur

- git config --global user.signingkey `<gpg-key-id>`

Ramasse miettes

- git gc

Il est bon de le faire de temps en temps, mais attentions à ne pas perdre de "dandling commits" !!!

Configuration d'un dépôt public

Avec le protocole git

A partir d'un repo local, avec le protocole **git** :

- git clone **--bare** src_folder dest_folder
- touch dest_folder/.git/git-daemon-export-ok

Ensuite, lancer le démon git

- git daemon

Avec le protocole http

Avec le protocole **http** :

- git clone **--bare** src_folder dest_folder
- git --bare update-server-info
- chmod a+x hooks/post-update

Les autres peuvent cloner comme ceci :

- git clone http://adresse/projet.git

Avec le protocole ssh

- git clone **--bare** src_folder dest_folder

Les autres peuvent cloner comme ceci :

- git clone user@server:chemin/projet.git

Créer une branche vide

```
git symbolic-ref HEAD refs/heads/nouvellebranche
```

```
rm .git/index
git clean -fdx
<travailler>
git add vos fichiers
git commit -m 'Premier commit'
```

Ecosystème

- gitosis
- gerrit
- gitblit
- ...

WORKING WITH THE COMMIT TREE

REBASE MERGE CHERRY-PICK STASH PATCH

REMOTING

TRACK BRANCH

COMMANDES

GIT GREP GIT CHECKOUT -- GIT RESET GIT REVERT GIT BLAME HOOKS Retrouver un commit perdu : git fsck —lost-found, puis git checkout/rebase/Cherry-pick/apply...